High Integrity Software Development (F29HD2):

A High Integrity Software Development Exercise and Comparative Study

Andrew Ireland
Department of Computer Science
School of Mathematical & Computer Sciences
Heriot-Watt University

January 15, 2008

1 Introduction

Safety should be the primary concern when building a railway network. The safety of a railway network typically depends upon the use of track-side signals in regulating the safe passage of trains. With the widespread use of software systems, railway signalling has achieved extremely high levels of reliability. The weakest link is typically the driver's response to track-side signals, *e.g.* drivers passing signals at danger. Within the railway industry such an event is called a SPAD (Signal Passed At Danger). During the month of February 2005, 23 SPADs occurred on the UK's railway network[1], of which 6 were classified as "serious". Note that February 2005 was a good month, *i.e.* there were 9 less SPADs than the average for the time of year. Although most SPADs do not result in accidents, they do represent a significant hazard. For instance, the Ladbroke Grove railway disaster[2] in which 31 people were killed resulted from a SPAD. Train protection systems exist that can prevent such disasters. Systems that are collectively known as *Automatic Train Protection* (ATP) ensure safe passage by monitoring a train's speed on the approach to track-side signals and activate the braking system automatically if necessary. Reports following the Ladbroke Grove disaster highlighted the need for train protection systems. But as yet, the majority of the UK railway network has little protection from SPADs.

The aim of this assignment is for you to implement the software component of a simple ATP system using the SPARK approach to high integrity Ada. You are provided with SPARK package specifications that define the safety-critical boundary of the system as well as a test harness written in Ada. Your task is to develop the system-critical control component as well as the implementation details of the boundary packages. In §2 a system-level description of the ATP system is provided. The software requirements of the exercise are detailed in §3, while the testing and formal proof requirements are outlined in §4. Finally, in §5 the deliverables that are expected of you are described.

In addition to the software development exercise outline above, you are also required to produce a comparative study, comparing the SPARK Approach with two other state-of-the-art approaches to developing formally verified software.

2 System-Level Description

The ATP system is located on board the train. It involves a central controller and 5 boundary subsystems that manage the sensors, speedometer, brakes, alarm, and a reset mechanism. The sensors are attached to the side of the train and detect information on the approach to track-side signals, *i.e.* they detect what the signal is displaying to the train driver. In order to reduce the effects of component failure 3 sensors are used. Each sensor generates a value in the range 0 to 3, where 0, 1 and 2 denote the *danger*, *caution* and *proceed* signals respectively. The meaning of these signals is as follows:

danger : a train must be stopped at a danger signal.

[1] Based upon Health and Safety Executive statistics.
[2] Occurred on the approach to Paddington station in London on 5 October 1999.

caution : a caution signal indicates that the next signal is currently at danger so the train's speed should be reduced in readiness to stop.

proceed : a proceed signal allows a train to continue on its way.

The fourth sensor value, *i.e.* 3, is generated if an *undefined* signal is detected, *e.g.* may correspond to noise between the signal and the sensor. The sensor value returned to the ATP controller is calculated as the majority of the 3 sensor readings. If there does not exist a majority then an undefined value is returned to the ATP controller.

If a proceed signal is returned to the ATP controller then no action is taken with respect to the train's brakes. If however a caution signal is returned to the ATP controller then the alarm is enabled within the driver's cab. Furthermore, once the alarm has been enabled, if the speed of the train is not observed to be decreasing then the ATP controller activates the train's braking system. In the case of a danger signal being returned to the ATP controller, the braking system is immediately activated and the alarm is enabled. Once enabled, the alarm is disabled if a proceed signal is subsequently returned to the ATP controller. Note that if the braking system is activated then the ATP controller ignores all sensor input until the system has been reset. If enabled, the reset mechanism deactivates the train's brakes and disables the alarm.

3 Software Requirements

The safety-critical software for the ATP system is to be written in SPARK. The safety-critical functionality is to be spread across 6 packages (subsystems) as described below:

`Sensors`**:** responsible for maintaining and providing access to the current values of the sensors.

`Speedo`**:** responsible for maintaining and providing access to the current value of the speedometer.

`Brakes`**:** provides control of the train's brakes and is responsible for maintaining the state information on the braking subsystem.

`Alarm`**:** provides control of the train's alarm and is responsible for maintaining the state information on the alarm subsystem.

`Reset`**:** provides control of the ATP reset mechanism and is responsible for maintaining the state information on the reset subsystem.

`ATP`**:** responsible for the overall control provided by the ATP system.

SPARK package specifications are given in appendix A for `Sensors`, `Speedo`, `Brakes`, `Alarm` and `Reset`. Note that the actual code for these package specifications can be obtained from:

`http://www.macs.hw.ac.uk/~air/hic/code/ATP/`

You are required to:

R1: Develop package bodies consistent with the specifications given in appendix A, *i.e.* you may modify the package specifications in order to improve information hiding. But any such changes should be made explicit via comments

R2: Develop a `ATP` package that implements the intended behaviour of the ATP controller as described in §2.

R3: The `ATP` package must be consistent with the 5 given package specifications as well as the test harness described below in §4.

R4: Using the SPARK proof tools, your SPARK code should be shown to be free from run-time exceptions.

4 Software Testing Requirements

You are given a test harness for the purposes of testing your ATP system software. The test harness is written in Ada and allows the simulation of the environment within which the ATP system is intended to operate. The test harness involves 3 packages that are described as follows:

`Env`: responsible for maintaining the state information associated with the `Sensors`, `Speedo` and `Reset` packages. The state information is read from a file called `env.dat` (see appendix B.1).

`Log`: responsible for logging the state information associated with `Sensors`, `Speedo`, `Alarm`, `Brakes`, `Reset` as well as the majority sensor reading. The logger writes to a file called `log.dat` (see appendix B.2).

`Test_ATP`: responsible for running tests using the subprograms within `Env` to step through the `env.dat` data file. The effects of the test data on the `ATP` controller are recorded in `log.dat` using the subprograms within `Log`.

Note that the code of these 3 packages is provided in:

`http://www.macs.hw.ac.uk/~air/hic/code/ATP/`

Example `env.dat` and `log.dat` files are also given in this directory.

5 Deliverables

Your submission **should** take the form of a report (hard-copy) and **must** include the following:

D1: A statement of any assumptions you have made about the requirements.

D2: A diagrammatic representation of the software architecture of your ATP system software. Your diagram should communicate the subsystems and their state, as well as their connectivity, *i.e.* imports and exports.

D3: The listing files generated by the SPARK Examiner for the 6 packages (specifications & bodies) that define the ATP system software (see requirements).

D4: The report file generated by the SPARK Examiner for all 6 packages.

D5: The `log.dat` file generated by your ATP system software for the `env.dat` file given in appendix B.1.

D6: The `log.dat` file generated by your ATP system software for the `env.dat` file given in:

`http://www.macs.hw.ac.uk/~air/hic/assignment/env.dat`

Note the above test data will not be available until Thursday February 28, 2008.

D7: Evidence that your SPARK code is free from run-time exceptions, *i.e.* the relevant `vcg` files together with `pogs` reports.

D8: A comparative study of the following three approaches to the development of formally verified software:

- The SPARK Approach
- ESC/Java2
- Spec#

Source material for each of these approaches can be found via the module web pages. Your study should outline each approach, highlighting strengthens and weaknesses. Use diagrams and examples where appropriate to illustrate key ideas and processes. Finally, you should compare and contrast the relative merits of each approach. Your report should be around 4000 words.

This assignment counts for 100% of the overall coursework mark for the module and should be submitted via the course work box located outside the Student Office (room 1.24) by **4pm on Friday 14 March 2008. Note that this is an individual project which means that your submission MUST be your own work.**

Appendix A: SPARK Package Specifications

Presented below are the SPARK package specifications for `Sensors`, `Speedo`, `Brakes`, `Alarm` and `Reset`.

A.1: sensors.ads

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       13/10/2006
--
-- Filename:            sensors.ads
--
-- Description:         Models the 3 sensors associated with the ATP system. Note that
--                      a single sensor reading is calculated using a majority vote
--                      algorithm.
package Sensors
   --# own State;
   --# initializes State;
is
   type Sensor_Type is (Proceed, Caution, Danger, Undef);
   subtype Sensor_Index_Type is Integer range 1..3;

   procedure Write_Sensors(Value_1, Value_2, Value_3: in Sensor_Type);
   --# global out State;
   --# derives State from Value_1, Value_2, Value_3;

   function Read_Sensor(Sensor_Index: in Sensor_Index_Type) return Sensor_Type;
   --# global in State;

   function Read_Sensor_Majority return Sensor_Type;
   --# global in State;

end Sensors;
```

A.2: speedo.ads

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       11/10/2006
--
-- Filename:            speedo.ads
--
-- Description:         Models the speedo device associated with the ATP system.
package Speedo
   --# own Speed;
   --# initializes Speed;
is
   subtype Speed_Type is Integer range 0..150;

   procedure Write_Speed(S: in Speed_Type);
   --# global out Speed;
   --# derives Speed from S;

   function Read_Speed return Speed_Type;
   --# global in Speed;
```

```
end Speedo;
```

A.3: alarm.ads

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       11/10/2006
--
-- Filename:            alarm.ads
--
-- Description:         Models the alarm device associated
--                      with the ATP controller.
package Alarm
   --# own State;
   --# initializes State;
is
   procedure Enable;
   --# global out State;
   --# derives State from ;

   procedure Disable;
   --# global out State;
   --# derives State from ;

   function Enabled return Boolean;
   --# global in State;

end Alarm;
```

A.4: brakes.ads

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       11/10/2006
--
-- Filename:            brakes.ads
--
-- Description:         Models the train braking subsystem associated
--                      with the ATP controller.
package Brakes
   --# own State;
   --# initializes State;
is
   procedure Activate;
   --# global out State;
   --# derives State from ;

   procedure Deactivate;
   --# global out State;
   --# derives State from ;

   function Activated return Boolean;
   --# global in State;

end Brakes;
```

A.5: reset.ads

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       11/10/2006
--
-- Filename:            reset.ads
--
-- Description:         Models the reset function of the ATP system that is required
--                      in order to disable the train's braking system.
package Reset
   --# own State;
   --# initializes State;
is
   procedure Enable;
   --# global out State;
   --# derives State from ;

   procedure Disable;
   --# global out State;
   --# derives State from ;

   function Enabled return Boolean;
   --# global in State;

end Reset;
```

Appendix B: Example Test Data & Results

B.1: An example env.dat file

The data within env.dat is used to simulate the values of the state variables associated with the boundary packages of the ATP system software. Each row represents the set of values at a particular point in time. The first row denotes the initial set of values in the test run while the last row denotes the final set of values. Columns 1 to 3 denote the values associated with the 3 sensors as described in §2. Column 4 gives the speed of the train while column 5 provides the state of the reset mechanism, *i.e.* where 0 denotes reset disabled and 1 denotes reset enabled.

```
0 0 0 50 0
0 0 0 55 0
1 1 1 56 0
1 1 1 55 0
1 1 1 54 0
1 2 2 59 0
2 0 2 60 1
2 1 1 66 0
0 0 0 67 0
0 1 0 69 0
```

B.2: An example log.dat file

The data within log.dat is generated by the logger (see Log). It records state and related information across a simulated test run of the ATP system software. The particular file shown below was generated using the env.dat file given in B.1. Note that there are double the number of entries in log.dat as there are in env.dat. This is because the logger is invoked twice within Test_ATP, *i.e.* before and after each invocation of ATP.Control.

SENSOR-1	SENSOR-2	SENSOR-3	MAJORITY	SPEED	ALARM	BRAKES	RESET
PROCEED	PROCEED	PROCEED	PROCEED	50	--	--	--
PROCEED	PROCEED	PROCEED	PROCEED	50	--	--	--
PROCEED	PROCEED	PROCEED	PROCEED	55	--	--	--
PROCEED	PROCEED	PROCEED	PROCEED	55	--	--	--
CAUTION	CAUTION	CAUTION	CAUTION	56	--	--	--
CAUTION	CAUTION	CAUTION	CAUTION	56	ON	--	--
CAUTION	CAUTION	CAUTION	CAUTION	55	ON	--	--
CAUTION	CAUTION	CAUTION	CAUTION	55	ON	--	--
CAUTION	CAUTION	CAUTION	CAUTION	54	ON	--	--
CAUTION	CAUTION	CAUTION	CAUTION	54	ON	--	--
CAUTION	DANGER	DANGER	DANGER	59	ON	--	--
CAUTION	DANGER	DANGER	DANGER	59	ON	ON	--
DANGER	PROCEED	DANGER	DANGER	60	ON	ON	ON
DANGER	PROCEED	DANGER	DANGER	60	--	--	ON
DANGER	CAUTION	CAUTION	CAUTION	66	--	--	--
DANGER	CAUTION	CAUTION	CAUTION	66	ON	--	--
PROCEED	PROCEED	PROCEED	PROCEED	67	ON	--	--
PROCEED	PROCEED	PROCEED	PROCEED	67	--	--	--
PROCEED	CAUTION	PROCEED	PROCEED	69	--	--	--
PROCEED	CAUTION	PROCEED	PROCEED	69	--	--	--